OFFICE OF THE CTO REPORT



Continuous API Sprawl

Challenges and Opportunities in an API-Driven Economy.

By Rajesh Narayanan, Mike Wiley

Table of Contents

Preface

The Application Programming Interface (API) economy is the totality of all public and private APIs that exist globally at any given moment. It is continuously expanding and will soon reach a point where it will become a driving force in the global economy. Just as the oil industry has dominated every aspect of our lives for over a century, APIs will become the core driver of the economy.

API SPRAWL IS HOW WE DESCRIBE BOTH THE EXPONENTIALLY LARGE NUMBER OF APIS BEING CREATED, AS WELL AS THE PHYSICAL SPREAD OF THE DISTRIBUTED INFRASTRUCTURE LOCATIONS WHERE THE APIS ARE DEPLOYED.

Prior to accelerated digital transformation, APIs were primarily viewed as a method of integration and a way to participate in a larger ecosystem. While, nearly half (43%) of businesses today already leverage APIs as a source of revenue[1] in addition to more traditional technical use cases, few have fully recognized the power of APIs to drive economic activity—although examples like Twilio and Stripe portend future use of APIs as a powerful business tool.

As we shift toward an API-driven economy, the problem we will need to contend with is the proliferation of API endpoints (a.k.a. API sprawl). Organizations that understand the cause of API sprawl and put into place a strong infrastructure—with people, processes, and tools to optimize their use of APIs—will thrive in this new API-driven economy.



Figure 1: API-driven economy

If data is the new oil, then APIs will become the new plastic. Responsible creation, use, and management of APIs will be critical, else the sprawl will pollute and wreak havoc on the ecosystem.

The first step in building a strong API infrastructure is getting a handle on API sprawl. While intuitively we might consider this as a potential future issue, it is not recognized as a significant business problem today—but it must be.

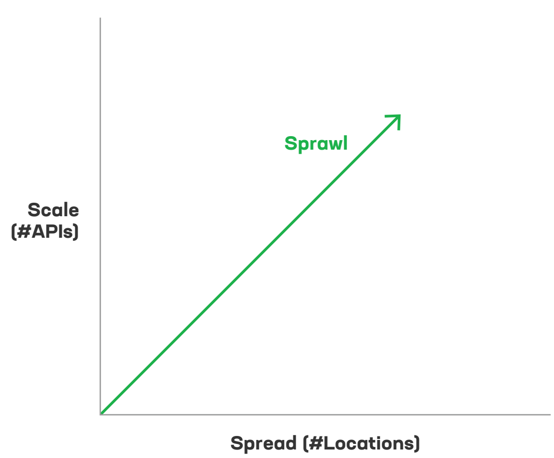


Figure 2: Sprawl is a function of scale and spread

Introduction

APIs are a contract between the service provider and service consumer. When any application uses an API, it needs to conform to an agreed-upon standard, with implicitly set expectations. What happens behind the scenes is of no concern to the consumer, enabling the service provider to use whatever means necessary to deliver the value. The service provider may choose any technology to deliver the service, and it may or may not optimize the resource being utilized to deliver the service.

ECONOMIC IMPACT

Beginning from a simple software construct for two systems to communicate without interdependency, APIs have evolved as a means for any entity connected to the internet to transact value through a well-defined contract.

Initially, APIs were primarily used as a standard means for two applications to talk to each other and exchange data. But APIs have evolved, and we are now seeing them used as a means for a service to deliver value to a consumer of the API.

APIs allow vendors to provide their goods and services in digital marketplaces. They have enabled applications to disrupt the transportation, hotel, and restaurant industries through ride-shares (Uber, Lyft, etc.), home rentals (Airbnb, etc.), and delivery services (DoorDash, etc.). Apps such as YouTube, TikTok, Instagram, etc., have empowered the creative among us be appreciated and rewarded by gathering a large fan following. APIs allow us to safely transact with anyone across the globe, securely, without needing to meet them in person. Internet of Things (IoT) devices, for example, prove their value by enabling their current state to be exchanged via an API.

From fun apps to lucrative enterprise applications, API-powered apps have permeated every aspect of our lives; and yet we are merely scratching the surface of the economic possibilities. To fully realize the economic and technical advantages of APIs we must first address a significant obstacle: API sprawl.

API SPRAWL IS CONTINUOUS AND GROWING

API sprawl happens when APIs become widely distributed without a holistic strategy that includes governance and best practices. Exacerbating this problem of distribution is the fact that API sprawl is also continuous as development teams follow a continuous application lifecycle process. Applications and APIs are constantly changing over time—and each version may spread differently than a previous version.

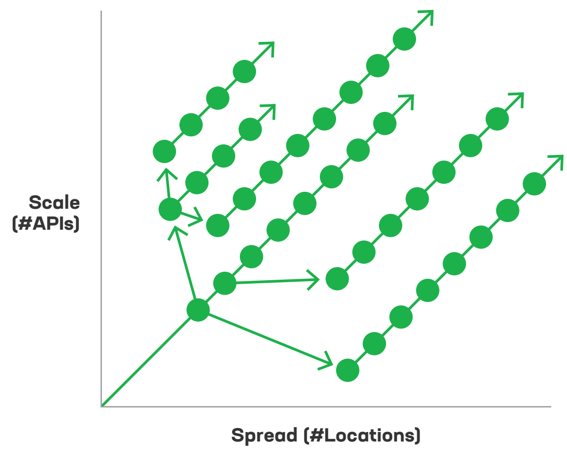


Figure 3: Even a single API can change and sprawl over time

Figure 3 shows how a single API can be the source of multiple versions for different purposes and each of those versions can then have their own version history.

It is important to recognize API sprawl as being continuous for a couple of reasons—(1) any market data estimate on the number of APIs is likely to be conservative, as the number of APIs will increase over time, and (2) any proposed solution must take into consideration this aspect of continuous growth.

APIS COME IN MANY SHAPES AND SIZES[2]

There are different classes and types of APIs. The internet is full of API definitions and classifications, but the one that aligns most closely with our own thoughts is the comprehensive list provided by ProgrammableWeb.com.

All APIs irrespective of type can be broadly classified as:

- **Public**—APIs available for anyone to use (e.g., Google Maps APIs).
- **Private**—APIs available only to internal teams or within an application cluster.
- **Partner**—APIs that integrate with a third-party vendor that can bring more value (e.g., an API that allows Netflix's app to be installed on a Roku device).

There are different types of APIs:

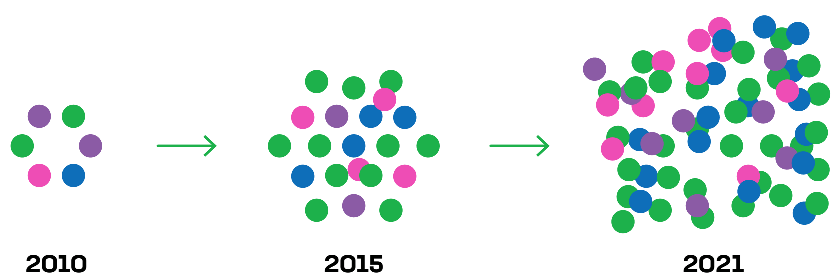
- **Web**—APIs that are accessible over the web.
- **Product**—APIs that are integrated into a product. When you buy the product and deploy it, a version of that product's APIs can be enabled.
- **Browser**—APIs in browsers that developers have access to and can recombine in different ways.
- **Standard**—APIs published by organizations or standards bodies for everyone to follow. For example, browsers could have a standard set of APIs to get access to the capabilities of the underlying system.
- **Embedded**—APIs that connect devices with proprietary data like IoT sensors.

APIs can also be characterized based on scope:

- **Single Purpose**—APIs with one single function (e.g., storage with Dropbox).
- **Aggregate**—APIs that aggregate services from similar companies and offer them as a single API (e.g., a storage aggregator may offer an API that aggregates storage across multiple storage providers by orchestrating with their proprietary APIs).
- **Microservices**—APIs that are typically within the scope of a microservices architecture. These APIs combine different microservices within an organization using some business logic, package them, and then expose them through a single API.

Figure 4 visualizes how APIs have developed over time from very few in monolithic systems to today where there are exponentially more APIs due to the proliferation of microservices architectures.

Figure 4: Visualizing API sprawl



Modeling API growth

Based on our estimation the number of APIs worldwide (public or private) is already approaching 200 million.

PARAMETERS

Quantifying API growth is not straightforward as there are several parameters that it depends on.

- Number of developers worldwide
- Number of developers writing APIs
- Number of APIs per developers per year
- Avg. growth in total # of developers
- Avg. growth in # of developers writing APIs
- Avg. growth in # APIs per dev per year
- Avg. shelf life of each API

Based on the above parameters we can reasonably derive a model to estimate API growth over a 10-year period. The table (Figure 5) shows the assumptions made in the data model to create the baseline. The blue line in the graphs below represents the baseline.

Figure 5: API sprawl baseline model parameters

1	#Devs worldwide in 2018	23.9
2	%Devs writing APIs	30%
3	Avg. #APIs per dev per year	9.4
4	Avg. worldwide growth rate of devs	NA
5	Avg. worldwide API dev growth rate	15%
6	Avg. growth in #APIs per dev per year	15%
7	Avg. shelf life of APIs in years	1

The API growth model assumes a 2018 start point of number of developers as 23.9 million. We were able to find references to number of developers from different sources (starting in 2018). To be fair, different sources may have numbers off by a wide margin; for example, SlashData estimates this number to be about 24M as of April 2021.[3]

Figure 6: Developer growth data

Year	# Devs (Millions)
2018	23.9
2019	26.4
2020	26.7
2021	27.1
2022	27.5
2023	27.7

cont.

Year	# Devs (Millions)
2024	28.7
2025	31.9
2026	33.7
2027	36.9
2028	40.3
2029	42.3
2030	45

For our model we have used 23.9 million in 2018. But as we shall understand later, this starting number doesn't matter much.

The table (Figure 6) shows the number of developers worldwide. The rows in black have an external reference, while the ones in red are estimated based on where we expect to be in 2030.

RESULTS

Figure 7 shows estimated API growth over a 10-year period. The model represents both very conservative (in purple) and aggressive (in light blue) growth. Irrespective of where the numbers land, we are witnessing a phenomenal growth in the number of APIs, resulting in potentially more than one billion APIs by 2031.

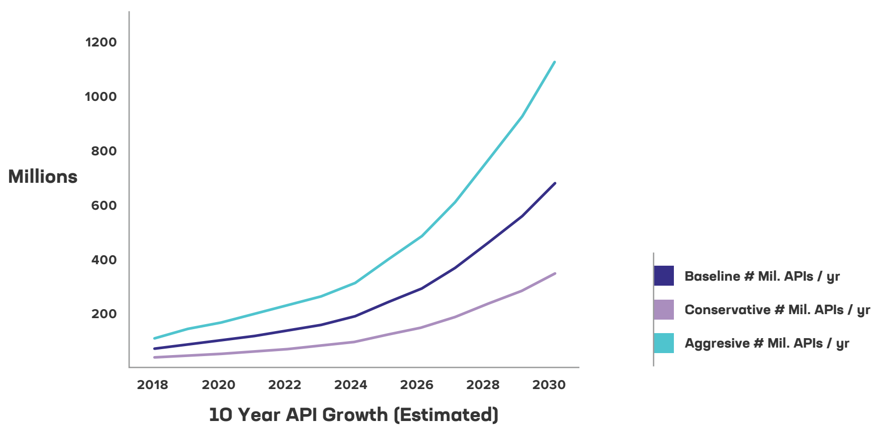
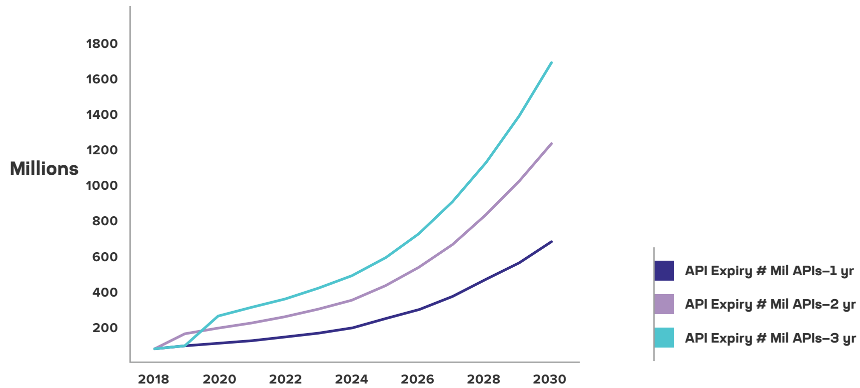


Figure 7: 10-Year estimated API growth

The 10-year growth estimate can be justified by the following analysis of the model.

API growth as a function of API shelf life: The 10-year estimated growth chart assumes a one-year shelf life. Figure 8 shows the API growth as a function of shelf life.

In this graph we estimate the number of active APIs in any given year. The graph cumulates the previous two and three years to estimate the number of active APIs at any given point in time, based on a two-year or three-year shelf life. According to this model, we will be approaching 1.7 billion active APIs by 2030.

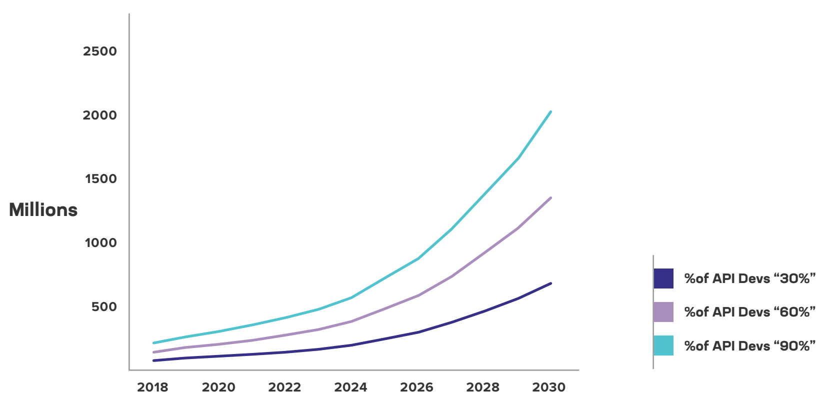


#Mil APIs as a function of API Shelf-life
(API expiry in yrs, i.e. #yrs API versions are supported)

Figure 8: API growth as a function of API shelf life

API growth as a function of the number of API developers: Another parameter affecting API growth is the number of developers. We see that the number of developers is growing over time and start with a baseline of 23.9 million developers in 2018.

We calculate API growth based on the number of developers who are developing APIs as 30%, 60%, and 90% of the worldwide developer pool. Figure 9 shows the growth of APIs as a function of increased developer pool.



#Mil APIs /yr as a function of Total #API Developers
(#API Devs = 20/60/90 % of total worldwide Devs)

Figure 9: API growth as a function of worldwide developers

Based on this model, we will be approaching 2 billion APIs by 2030 with just a one-year shelf life.

Summarizing the API growth model, we can say that it truly does not matter what our starting point is. Whether we assume 24 million developers in 2018 or 2021, the number of APIs by 2030 will be in the 100s of millions, making it a significant scalability, manageability, and security challenge for our customers and the industry. It does not matter what parameters of the model we tweak; API sprawl will be a global problem. Discovery, networking, integration, and security are set to become significant challenges for the entire Dev and Ops ecosystem.

Contributing Factors to API Sprawl[4]

All business units deploy services through some application platform, and regardless of whether the application is monolithic or microservices-based, APIs are becoming the standard means to interact with these services.

LACK OF STANDARDS

While there are data-formatting standards, such as the use of JSON and XML to encapsulate the data exchanged by APIs, there are few global standards for APIs themselves.

Interestingly, the few emerging API standards that exist tend to focus on business domains. For example, FDX is an organization “dedicated to unifying the financial services ecosystem around a common, interoperable, and royalty-free technical standard… aptly named the FDX Application Programming Interface (FDX API).”[5] Its membership includes nearly every financial organization, making it a powerful force for establishing a standard.

Within the technology domain, however, APIs have followed their natural predecessors, the Command Line Interface (CLI). No two are the same. For example, different Infrastructure as a Service (IaaS) providers can represent a set of server resources as *Server Pool* or *Server Farm*. The two APIs might be doing the same thing, resulting in significant overlaps in the industry.

APIs may be reflective of an application’s internal data model and not follow an established or formal standard. The lack of a common shared model contributes to API sprawl.

MICROSERVICES ARCHITECTURE

As businesses go through digital transformation, they are increasingly adopting cloud-based microservices architectures. The move to microservices results in an application being composed of many dozens of APIs. In addition, organizations tend to create an access layer to legacy systems via APIs.

Microserices are accretive and often used to extend “traditional” apps to hidden interfaces via APIs. This is actually a considerable source of API sprawl, because the APIs are both northbound to interfaces via microservices and horizontally between microservices.

CONTINUOUS SOFTWARE DEVELOPMENT

Adding to this issue is the state of modern software development, which is itself continuous and often results in multiple versions of the same API as mentioned earlier.

As enterprises increasingly adopt the continuous software lifecycle, developers can churn out many APIs, and many versions of an API, over a short period of time. This can make the API

versions hard to track. In addition, documentation suffers if developers are not meticulous in their practice. Thus, continually modifying, updating, and changing APIs amplifies the sprawl (e.g., enterprises may have different versions of the same API available in different regions).

The agile development process enables teams working on short sprint cycles to develop new APIs and enhance existing ones rapidly. Keeping track of different versions of an API that has been deprecated, deployed, or deleted can become a daunting task to manage. Enterprises may need to maintain deprecated APIs over a longer term due to existing customer support issues. If there are changes to the development team or the operations team, these may turn into zombie APIs running somewhere in the infrastructure but not managed by any given team.

The resulting state of APIs, with multiple versions due to both continuous development and a lack of standards, poses a significant challenge to integration efforts.

INTEGRATION CHALLENGES

Integration is a main focus for developers today and hence a significant factor in API growth. App modernization efforts drive 58% of organizations to add APIs as ways to connect modern user experiences with existing, traditional applications.[6] According to the Postman blog,[7] ~70% of enterprises cited integration between internal applications, programs, or systems as the main reason to create new APIs.

Many companies have internal initiatives to create seamless integration between their software assets. Such integration efforts are complex orchestrations of technical and organizational challenges. In any enterprise, most of these assets either develop organically or come in through mergers and acquisitions.

But more than acting just as application "glue," APIs are now treated as "business glue" because they enable participation in economic and service ecosystems to strengthen strategic partnerships. 83% of organizations today consider API integration a critical part of their business strategy.[8]

The use of APIs creates a well-defined interface between two apps to integrate and enables the organization to preserve the autonomy of different product teams serving internal and external customers. Care must be taken to avoid the duplication and confusion that can come from distinct business and product units operating autonomously and integrating with APIs.

SILOED BUSINESSES UNITS

Organizational business units are siloed by design. Different product and dev teams also become siloed depending on the best practices they adopt. Integrating the services created by these business units can be a daunting challenge.

Within a well-established business unit, the enterprise may have multiple product teams developing separate microservices and products for an uber-project. In addition, mergers and acquisitions introduce new silos, new architectures, and new APIs. Teams can end up reinventing APIs for the same service repeatedly, which can result in integration challenges later. Teams will jostle and struggle with questions like, "Which API from which team is better?" The discussion can quickly descend into organizational politics and not-invented-here[9] syndrome.

HYBRID INFRASTRUCTURE

An outcome of siloed business units is that every team tends to adopt an infrastructure strategy (Figure 10) they are comfortable with, based on familiarity and skill sets. On-prem teams spar over OpenStack vs. VMware and other home-grown technologies as they move to the cloud it becomes GCP vs. AWS vs. Azure vs. whatever they are most acquainted with. APIs thus get dispersed over many locations and become difficult to track.

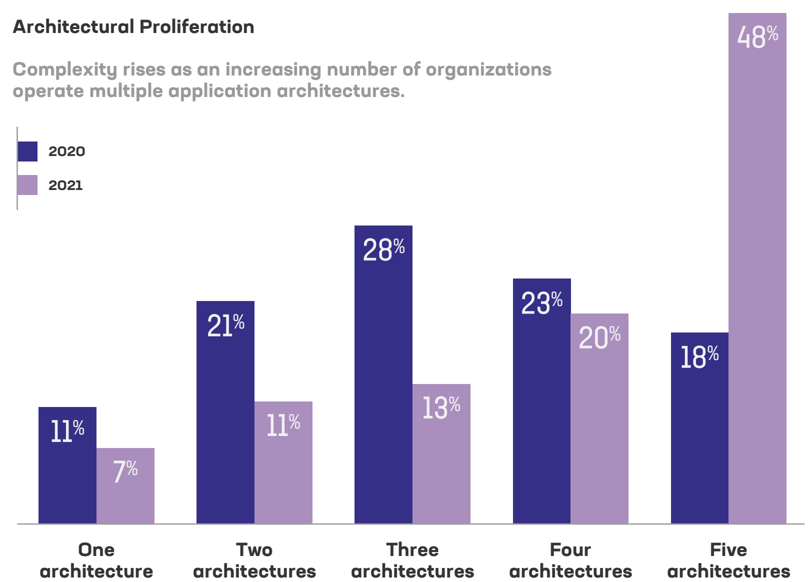


Figure 10: Organizations operating multiple architectures[10]

As the definition of multi-cloud expands (Figure 11) to include emerging edge computing platforms and environments, API sprawl will continue to expand as well.

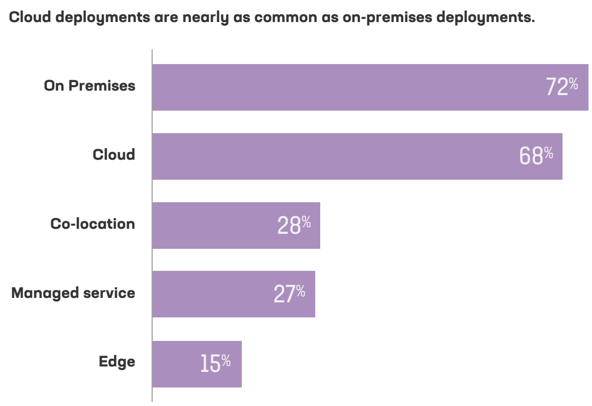


Figure 11: Cloud deployments as opposed to others

EDGE COMPUTING

As the cloud expands to wherever the assets become available, and closer to where the data is generated, edge computing becomes part of a distributed cloud.

API mobility: APIs have become the primary means to interact with different data sources. APIs are also moving closer to the data to collect, collate, and pre-process it. If the data source moves, as in the case of a mobile or IoT device, the related API also becomes mobile. An API could move to a new destination that may be outside the geo-fenced location with constraints. Different versions of an API might be needed depending on the type of enterprise accessing it, physical location, security needs, or compliance with regulatory environments.

Data sprawl: Data sprawl contributes to API sprawl because data by its nature is dispersed. As APIs become the gateway to data, an outcome of edge computing is that APIs get distributed to where the data is located, which adds to the sprawl. App developers use APIs to gather that data and create further value—but also further complexity.

EVERYTHING-AS-A-SERVICE

The next evolution from Software as a Service is Everything as a Service (XaaS), where anything tangible can now be consumed in an as-a-service model. With software, anything tangible is modeled as a “digital twin.”[11] The scheduling and delivery of these things is through an API. Companies like Airbnb, Uber, DoorDash, etc. are all examples of XaaS.

As the number of XaaS offerings increases, we are back to the need for standards, or some common way to represent these APIs. But unification of these APIs under some larger umbrella is impractical. We are left with a significant challenge, but also an opportunity.

Why is this a problem today?

In a digital world built on the API economy, those APIs must be 100% reliable. One must be able to access them anytime from any location, device, or entity.

From the Postman blog,[12] enterprises list the top four reasons when choosing an API as reliability, security, performance, and documentation, in that order.

API Academy[13] outlines several factors contributing to reliability: consistency, availability, low latency, security, and status reporting. The question is "How can one reliably track reliability when the APIs are sprawled across a heterogeneous and distributed cloud?"

There are several factors affecting the reliability of APIs that are exacerbated due to the immense scale and spread created by API sprawl.

OPERATING AT SCALE

In a constrained environment (e.g., within a cluster), it is easier to discover, connect, secure, and establish trust through centralized management and control. Sprawl forces us to adjust the way we think—from a purely hierarchical model to a distributed and autonomously scaling model.

No source of truth

As the number of APIs and the complexity of the apps grow with the organization, it becomes very hard to track where these apps are located. All the APIs may not be registered or discoverable as they might be behind different infrastructures.

Whatever the real numbers are, the problem is massive. APIs have a shelf life and become unsupported if ignored by the developers. We will need an inventory of deprecated or unsupported APIs—something like an "API garbage collector."

Within the decade we will see services mushroom, which will validate public or private APIs for the latest versions, supportability, security etc., and offer it as a SaaS platform: a source of API truth.

API discovery challenges

Another significant challenge enterprises are already facing is discovering APIs within and outside the enterprise. Existing approaches only tend to cover within an application cluster (e.g., API gateways within a service mesh). But even within a single enterprise there could be hundreds of clusters using different service mesh technologies.

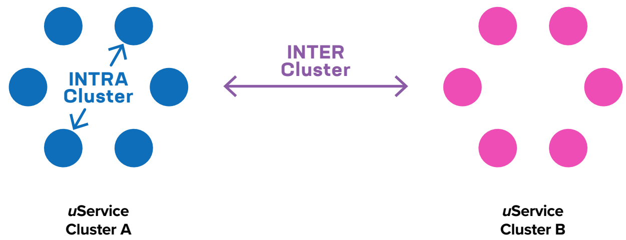


Figure 12: Intra-cluster vs. inter-cluster

APIs are not just intra-cluster, but also inter-cluster (Figure 12).

Versioning and documentation

The problem with versioning is primarily when APIs change rapidly due to updates, being deprecated, or not supporting certain protocols. The expectation is that the remote service calling this API also needs to change. When the microservices are being designed by the same team for the same application, this may work, but the complexity rapidly increases if the APIs are being published as third party or being consumed from a third party.

APIs also have a lifetime and may become unsupported or unavailable over time. If an API fails, the backup must be implemented by the developer who must also make changes to their application to handle responses.

Connectivity challenges

Approaches like service mesh assume that robust and reliable network connectivity already exists. Within an enterprise this may be true to an extent. For a finite number of high-priority projects the network infrastructure and security team can become closely involved in network planning, configuration, and security to ensure reliable end-to-end connectivity is available.

In many cases when the APIs are across clusters (public or private), a simple means to connect these APIs may not be available with conventional or legacy networking approaches.

SECURITY AT SCALE

More than nine out of ten of enterprises experienced an API security incident in 2020.[14] Every API thus becomes a point on the security perimeter that can be potentially compromised if not properly architected or protected.

To reiterate, the term “sprawl” is an indication of both numbers in terms of scale and being physically dispersed over a wide area. At such scale, security cannot be implemented as an add on feature. It must become part of the entire API lifecycle, from code to deployment to end-of-life.

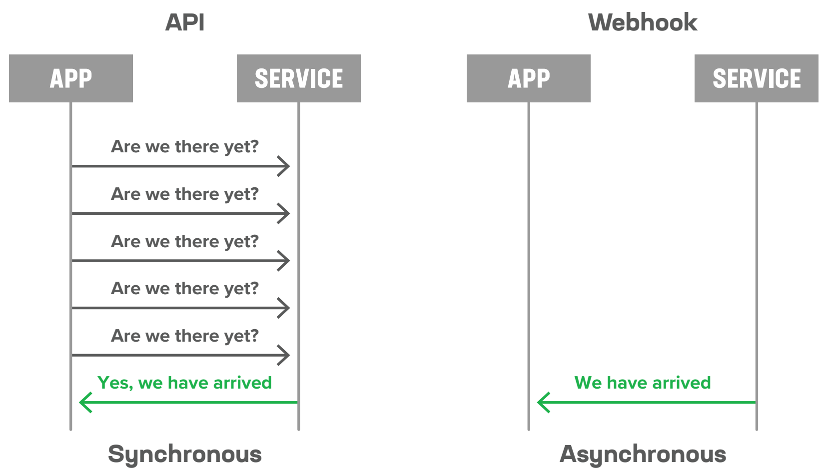


Figure 13: APIs and webhooks

Figure 13 shows the difference between APIs and webhooks (addressed below).

APIs prone to fraud and malicious behaviors

If enterprises are not careful when using APIs, there are many opportunities for fraud and malicious behavior to creep into an implementation. Product and dev teams may be on a deadline looking to incorporate a certain feature provided by an external API. If due diligence is not performed on the API provider, it could result in basic security issues and sophisticated attacks, such as tainted data meant to undermine a business.

Assume we have two competing enterprises: RED and BLUE. Both are a type of retail or brick-and-mortar store. Now BLUE has implemented an app showing the nearest location of its different stores. The map service used by BLUE is provided by PURPLE via an API.

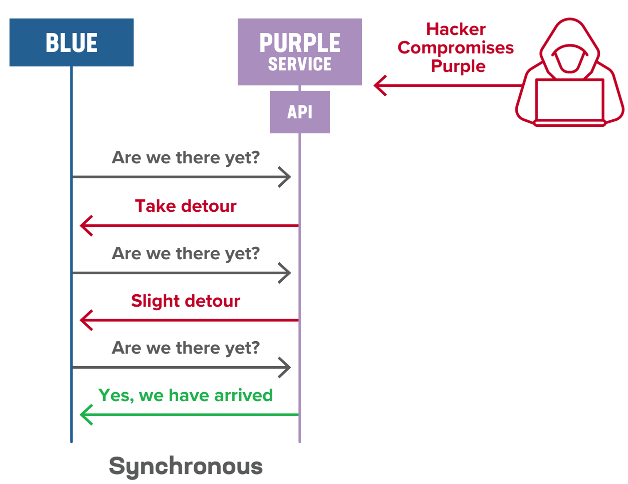


Figure 14: API fraud example

Figure 14 is an example of how a malicious actor does not need to hijack the client (BLUE) to cause long-term problems for the client. RED acquires or invests in PURPLE and inserts

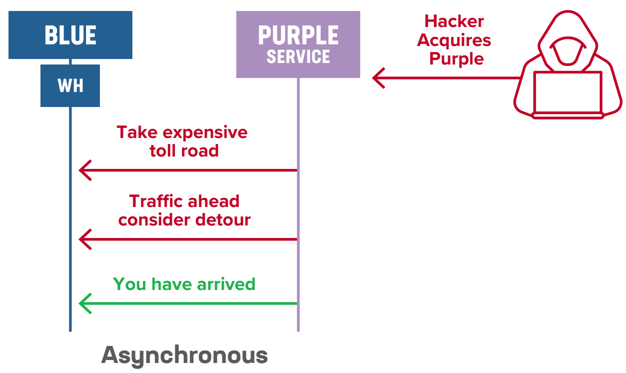
randomness or inaccuracies into the API only when BLUE accesses it. A more complex and long-term scenario is that state actors or a hacker group hijacks PURPLE's APIs to affect BLUE's performance and eventually affects BLUE's stock; this could take much longer to detect or might even go completely undetected.

Webhooks can be weaponized

Webhooks[15] are basically user-defined HTTP callbacks or small code snippets linked to a web application. These callbacks are triggered by specific events in a remote site which use these callbacks. For example, if the BLUE service has registered a webhook with PURPLE service, it is indicating to the PURPLE service to call it when there is a specific triggering event.

Unlike APIs, webhooks are asynchronous. Any application needing an event notification from another software program can register a URL. Most security teams seem to rely on the anonymity of a webhook URL to maintain its security.

Figure 15: Webhooks fraud example



Webhooks are potentially more dangerous (Figure 15) as a malicious actor can directly call BLUE if they can compromise PURPLE. There is no need to wait for BLUE to initiate the request.

Once a webhook URL is exposed and its data-model revealed, any hacker can hack into a service and send a malformed data-object asynchronously. A Dark Reading article[16] exposed how Slack's webhooks could be weaponized by creating phishing attacks on a Slack channel that was compromised due to an exposed webhook. "Graves said a quick scan of GitHub had thrown up more than 130,000 public code results that contained Slack webhook URLs, most of them containing the full unique value."

A hacker can easily scan all the GitHub repos for public Slack webhook URLs and automate the entire process of sending out malformed messages with phishing links.

TRUST DECAYS

API security is a complex subject and “trust” makes up an important component.

When a service within the enterprise accesses a well-known external API, the platform must implement a mechanism by which the calling service can be assured of the accuracy of the received response from the external API. Just because the API response looks valid, and comes from a previously validated endpoint, does not mean it can be trusted (Figure 16).

Figure 16: Proof of trust is complicated

Proof of Identity + Proof of Work
≠
Proof of Trust

Either the response could be inaccurate due to quality issues, or as we learned earlier, inaccuracies can be explicitly inserted to make the business less competitive. Like the agile process, trust is continuous and must be constantly validated.

SECRETS SPRAWL

Secrets[17] are anything allowing privileged access into a system. There are many types of secrets in computer systems: username/password combinations, client ID/secret, certificates, tokens, keys, and database credentials.

The most common method for APIs to authenticate are through API keys. Customers can have API tokens distributed randomly in unsecure locations. The API keys could be stored in clear-text within a database or source code (git-repos), available in an office email or personal email, or found in a backup drive or Google drive.

This is where the API sprawl increases the security risks. Many attack vectors get exposed when secrets are spread out across a distributed infrastructure. Hard-coded keys or credentials (for instance in git-repos) can unintentionally expose an application service. It takes only one API key to be compromised for an attacker to gain access to critical infrastructure.

There are two key takeaways supporting the above assertions found in the *2021 IBM Security X-Force Cloud Threat Landscape Report:* (A) Two-thirds of cloud incidents were related to misconfigured API keys allowing improper access, and (B) API credential exposure through public code repositories frequently accounted for threat actor access into cloud environments.

Unmanaged API sprawl is a security breach waiting to happen.

What can we do about it?

API sprawl is an uncomfortable outcome of modern software architecture. We can reasonably conclude that API sprawl is here to stay, and we must deal with it in a practical and scalable manner.

Ultimately, the solution space consists of both intra-cluster and inter-cluster operational complexities. We use the term “cluster” loosely to represent applications delivered through any software architecture—from monolithic to microservices. These clusters can be within an organization, or across multiple enterprises that need to connect their apps.

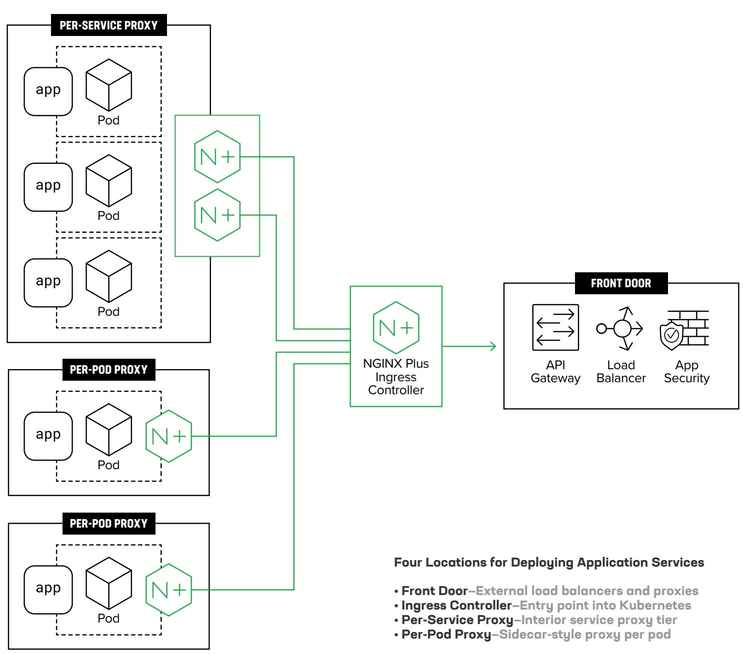
INTRA-CLUSTER DISCUSSION

Intra-cluster API architectures are well-known constrained environments. An enterprise will have hundreds if not thousands of independent and autonomous clusters operating simultaneously—from legacy and web 2.0 to microservices architectures.

Product and dev teams will have adopted certain frameworks to use based on their existing requirements. Enterprises may acquire or use third-party software, which may use a different software architecture. Irrespective, each team has their own set of opinionated software stacks and there is no single standard of implementation in any enterprise (see Figure 10: Organizations operating multiple architectures).

But most enterprises are evolving toward a microservices-based architecture. So let us understand intra-cluster architectures within the context of microservices (Figure 17).

Figure 17: Microservices deployment with API and Ingress controller[18]



API gateways[19]**:** An API gateway is used for application routing, rate limiting, security, request and response handling, and other application-related tasks. Say you have an application in which the requested information needs to be collected from multiple services. API gateway distributes the user requests to different services, gathers the responses from all microservices, and prepares the final response to be sent to the user.

While the role of API gateways has come to be more recognized within the context of microservices, there is no reason they cannot exist independently on the network.

Ingress controllers: An Ingress controller is little more than an API gateway. Its primary function is to perform L7 routing, which in today's deployment architectures becomes API routing, so it ends up being called an API gateway.

API gateways have arrogated scaling, security, and other connection-related tasks, including load balancing, SSL termination, firewalling, offload, etc. The difference is that an Ingress controller is a component of Kubernetes (K8s), whereas an API gateway is more of a stand-alone architecture solution.

Sidecar proxy: Service mesh has already become the most popular means to implement microservices. With the addition of a service mesh sidecar proxy, the architects at Lyft identified the key problem areas[20] with microservice architectures: simplifying connectivity, discoverability, and security within a cluster.

While service mesh has evolved to address many challenges of API discovery, connectivity, and security, the API sprawl problem is just too massive and distributed. Even for a mid-size enterprise API sprawl can be a daunting problem that cannot be solved using intra-cluster technologies, like API gateways, Ingress controllers, sidecar proxies, etc.

Solutions developed for intra-cluster are not wrong. The technology has been optimized to serve a specific purpose and is being widely adopted. There are many approaches to intra-cluster API discovery, connectivity, and security.

Individual teams may also use the microservice technologies of their choice and can get opinionated on why their chosen tech has a better approach to implementing API discovery, connectivity, and security. It is unreasonable and impractical expect them to adopt a different implementation, even if there is a corporate mandate.

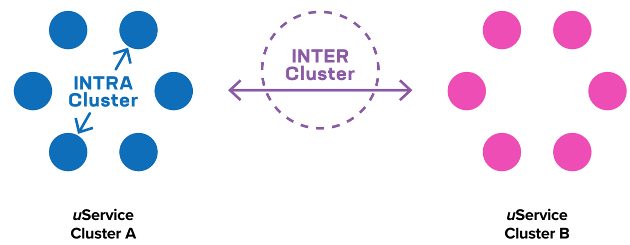
API SPRAWL IS AN INTER-CLUSTER PROBLEM

Whether public or private, the scope is inter-cluster, so it might be best to scope the problem as well to inter-cluster issues.

Solution scope

We believe the solution should hence involve an intermediary (proxy) device focused on solving inter-cluster connectivity, security, and integration challenges.

Figure 18: Intra-cluster vs. inter-cluster



This focused scope has several advantages. Primarily, it is least disruptive architecturally as it does not require developers to change or modify their dev environments. Security is also simplified, as existing API gateways can continue to operate in their current manner. This approach scales to any number of clusters and does not care whether the APIs are on prem, in the cloud, or are provided by a third party.

Solution requirements

These requirements are derived from the problems created by API sprawl.

- **Provide a single source of truth:** Solution provides a means to discover and access all the APIs approved for use by the enterprise.
- **Enable seamless API discovery:** Solution provides a means to discover new APIs.
- **Ensure proper versioning and documentation:** Solution ensures the latest state of the API documentation is available.
- **Enable API-to-API connectivity:** Solution ensures APIs to connect even when there are no direct routes available between the two.
- **Preserve security at scale:** Solution ensures that the complexity of API security does not increase non-linearly with the number of APIs
- **Monitor API reliability:** Solution ensures reliability through consistent and uniform monitoring across the deployment.
- **Provide trust as a metric:** As APIs and webhooks can be abused, the impact may go well beyond security with the potential to shake the foundations of a business using it. Enterprises should be able to trust the APIs they are using beyond just identity.

Solution approach: API gateway 2.0

A solution to effectively deal with API sprawl demands a fresh approach and potentially a new intermediary device. For this discussion, we'll call it the API gateway 2.0 (AGW 2.0). This new API gateway needs to evolve to address the requirements of challenges created by API sprawl.

Evolution of the "proxy": If we review the history of intermediary devices in the network, two fundamental devices emerge: the forward proxy and the reverse proxy.

Forward proxy: A forward proxy (Figure 19) acts as an intermediary to protect the internal application by transacting with the remove server from a client's behalf. A VPN server utilizes the concept of a forward proxy and adds encryption as an add-on feature. You want to browse Netflix on your corporate network, but it is blocked. You can connect to a VPN server and forward your traffic to Netflix.

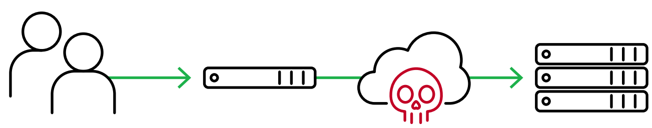


Figure 19: Forward proxy

Reverse proxy: Is where it acts as an intermediary from an external untrusted network that does not need to know the exact destination of the packet on the internal network (Figure 20). Load balancers, firewalls, and API gateways are all examples of a reverse proxy.

Over the last three decades we have observed the reverse proxy add on many layers of functionality. In the last 10 years, system-design approaches have all relied on application scaling by adding session management functionality to the reverse proxy.

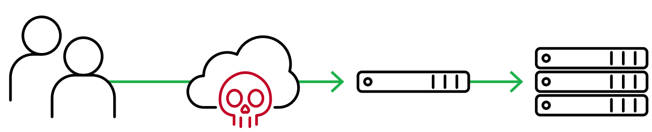


Figure 20: Reverse proxy

Why we need a different approach: Neither the forward proxy nor reverse proxies are designed to solve the problems of massive API sprawl. Let us take for example a simple case where there are two applications in two separate networks, each behind four layers of firewalls.

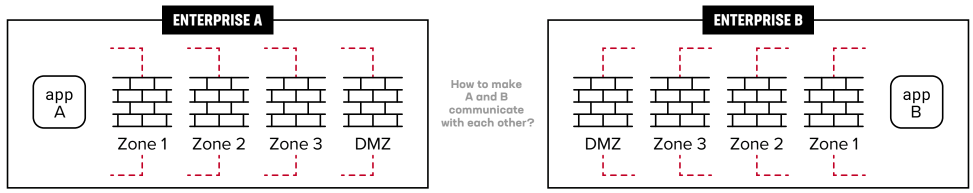


Figure 21: Challenges with basic app-to-app communication

This may work for a single case as discussed earlier. Different infrastructure teams can collaborate to make the connectivity happen. However, when we have billions of API endpoint conversations waiting to happen, in a secure, safe, and reliable manner, existing solutions are impractical. Even ensuring simple low-bandwidth and high-latency reliable connectivity will be next to impossible. These connections need to be dynamically set up, ephemeral, and secure. This complexity at scale demands a different approach.

Bidirectional (a.k.a. meet-me) proxy: We believe a different type of proxy is needed. This is not a new concept and has been applied before but has been primarily used in solutions providing multi-media communications (e.g., many conferencing apps have a client that dials out of the enterprise and connects to a meet-me point). However, there are very few examples of a bidirectional proxy being used as a tool of choice for enterprise-grade application-level connectivity and security—yet.

We reintroduce the bidirectional or meet-me proxy. The primary assumption in making a meet-me proxy work is that all endpoints (devices, apps, humans, etc.) can route traffic to it in some manner.

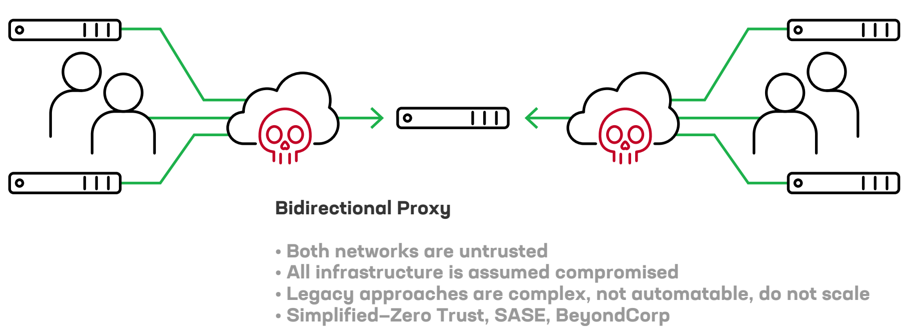


Figure 22: Bidirectional proxy

There are several advantages of a bidirectional proxy.

Basic network bridge: If all devices and apps needing to communicate with each other can connect to the meet-me proxy, then it can work as a basic network bridge.

Non-disruptive: It does not need to be installed in the local network, so network teams do not have to learn yet another device to manage.

Can be built on any layer: Figure 22 is only a concept. The system can be lightweight or a heavy bare-metal appliance as needed, while the proxy can be a lightweight container connecting a few API endpoints, or a managed networking device in a colo facility.

Technology exists: Probably the most important for the community is we do not have to invent a new technology. The building blocks for this already exist and—depending on the requirements of an enterprise—it is possible to build such a system today. We believe a meet-me proxy provides the most elegant and simple solution to address the API sprawl problem.

Summary

We are headed toward an API-driven economy, and even if you are not participating in it now, you will be soon. APIs are the fundamental step toward digital transformation irrespective of where you are in that journey. You may not want to address API sprawl at this moment, but you can't ignore it.

It is vital for all enterprises to recognize and be aware of the complexities in managing their API strategy. While an organization is trying to address this up the chain and create strategic insights, it is important to understand how you are using your APIs today. Product and engineering teams must appreciate the impending strategic shift now to be nimble, avoid business tensions, and effectively compete and gain a first-mover advantage.

F5 is dedicated to its customers and are partners in your business transformation journey. We are also committed to working with an ecosystem of partners, vendors, standards bodies, and open-source communities to ensure a robust, reliable, and safe infrastructure supports the new API-driven economy at global scale.

References

[1] https://offers.cloud-elements.com/2020-state-of-api-integration-report

[2] This section is thanks to the https://programmableweb.com and their YouTube channel. The specific video with the definitions can be found here.

[3] https://www.theregister.com/2021/04/26/report_developers_slashdata

[4] https://betanews.com/2021/01/13/securing-modern-apps-api-sprawl/

[5] https://financialdataexchange.org/

[6] https://www.f5.com/state-of-application-strategy-report

[7] https://www.postman.com/state-of-api/api-strategies/#api-strategies

[8] https://offers.cloud-elements.com/2020-state-of-api-integration-report

[9] https://www.bmc.com/blogs/not-invented-here-syndrome/

[10] https://www.f5.com/state-of-application-strategy-report

[11] https://en.wikipedia.org/wiki/Digital_twin

[12] https://www.postman.com/state-of-api/api-strategies/#api-strategies

[13] https://apiacademy.co/2021/06/continuous-monitoring-for-api-reliability/

[14] https://www.techrepublic.com/article/91-of-enterprise-pros-experienced-an-api-security-incident-in-2020/

[15] chargebee.com/blog/what-are-webhooks-explained/

[16] https://www.darkreading.com/cloud/slack-s-incoming-webhooks-can-be-weaponized-in-phishing-attacks

[17] https://blogs.ultima.com/windows-environment-security

[18] https://www.nginx.com/blog/deploying-application-services-in-kubernetes-part-2/

[19] https://stackoverflow.com/questions/59071842/ingress-controller-vs-api-gateway

[20] https://glasnostic.com/blog/service-mesh-istio-limits-and-benefits-part-1

DC1021 | WP-X-PILLAR-737943292